

Unisenac
Campus Pelotas



Algoritmos e Estruturas de Dados I

CENTRO UNIVERSITÁRIO UNISENAC – CAMPUS PELOTAS
CURSOS SUPERIORES: ESCOLA DE TECNOLOGIA
PROF. EDÉCIO FERNANDO IEPSSEN

Dicionários

pythonacademy.com.br/blog/dicts-ou-dicionarios-no-python

Introdução

Os dicionários são coleções de itens e seus elementos são armazenados de forma não ordenada.

Seus elementos contêm uma chave e valor, isto é:

- Uma `chave` que vai servir para indexar (posicionar) determinado elemento no dicionário.
- Um `valor` que contém... Bem, um valor 😊 Este valor aceita diversos tipos: listas, outros dicionários, inteiros, strings e etc.

Sua sintaxe básica é: `{'chave': 'valor'}`. Utiliza-se `{}` para delimitar o dicionário e a chave é separada do valor por dois pontos `:`.

Exemplo de sua sintaxe:

```
1 dicio = {'chave': 'valor'}
2
3 print(type(dicio))
```

Quando utilizar `type()` e a saída for essa abaixo, pode ter certeza que é um dicionário!


Resultado

```
<class 'dict'>
```

<https://pythonacademy.com.br/blog/dicts-ou-dicionarios-no-python>

C: > algo1_23_2 > manha > dicionarios > exemplo.py > ...

```
1  # listas / vetores / arrays
2  produtos = []
3
4  # tuplas (de certa forma, semelhantes as listas-porém as tuplas são "imutáveis")
5  alunos = (12, 20)
6
7  # dicionários / objetos
8  agendas = {}
9  agendas = dict()
10
11 # exemplos de dicionários
12 contatos = {"Ana": "99101.0203",
13             "Bianca": "98420.3040",
14             "Carlos": "99912.3456",
15             "Débora": "99130.3132"}
16
17 print(contatos)
18 print(contatos["Bianca"])
```

C: > algo1_23_2 > manha > dicionarios >  exemplo.py > ...

```
19
20     # alterar o conteúdo do dicionário
21     contatos["José"] = "99950.6070"
22     contatos.update({"Eduardo": "99244.5566"})
23
24     print(contatos)
25
26     # formas de percorrer as chaves do dicionário
27     for nome in contatos.keys():
28         | print(nome)
29
30     # formas de percorrer os conteúdos do dicionário
31     for fone in contatos.values():
32         | print(fone)
33
34     # formas de percorrer as chaves e conteúdos do dicionário
35     for (nome, fone) in contatos.items():
36         | print(f"{nome} - {fone}")
```

```
# Principal aplicação: Listas de dicionários
clientes = [
    {"nome": "Luis Carlos", "idade": 25},
    {"nome": "Ricardo José Costa", "idade": 29},
    {"nome": "Ana Santos", "idade": 51},
    {"nome": "Bianca Souza", "idade": 40},
    {"nome": "Marcos Pereira", "idade": 20},
]

print("-----")
# listar os dados
for cliente in clientes:
    print(f"{cliente['nome']} - {cliente['idade']} anos")

# ordenar os elementos da lista
# lambda: palavra reservada do Python para declarar uma função anônima
clientes2 = sorted(clientes, key=lambda cliente: cliente["nome"])

print("=====")
# listar os dados
for cliente in clientes2:
    print(f"{cliente['nome']} - {cliente['idade']} anos")
```

Function object

Stores the result of the expression

Arguments

One or multiple arguments, separated by a comma

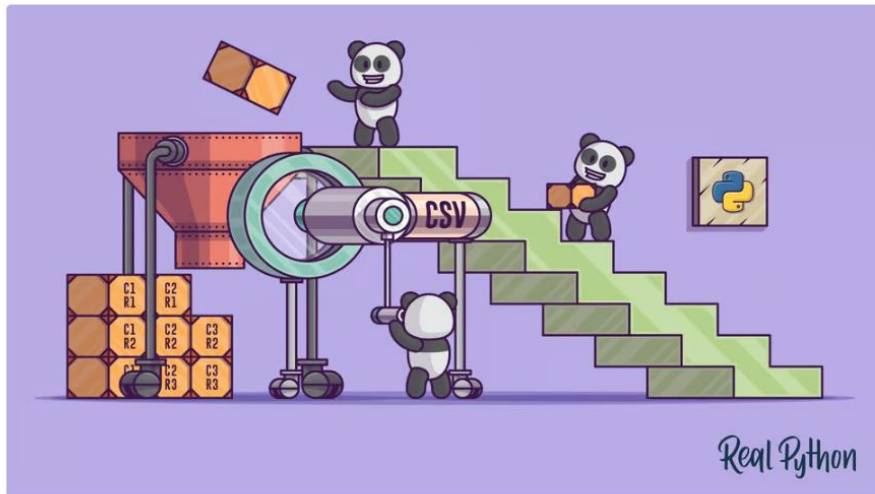
`func = lambda x, y: x + y`

Keyword

Used to define a lambda function

Expression

Single expression to evaluate and return the resulting value



Reading and Writing CSV Files in Python

by Jon Fincher  85 Comments

 data-science  intermediate

Mark as Completed



 Share

 Share

 Email

Vamos ser sinceros: você precisa inserir e retirar informações de seus programas por meio de mais do que apenas o teclado e o console. A troca de informações por meio de arquivos de texto é uma forma comum de compartilhar informações entre programas. Um dos formatos mais populares para troca de dados é o formato CSV. Mas como você usa isso?

Vamos deixar uma coisa bem clara: você não precisa (e não vai) construir seu próprio analisador CSV do zero. Existem várias bibliotecas perfeitamente aceitáveis que você pode usar. A [csvbiblioteca](#) Python funcionará na maioria dos casos. Se o seu trabalho requer muitos dados ou análise numérica, a [pandasbiblioteca](#) também possui recursos de análise CSV, que devem cuidar do resto.

Neste artigo, você aprenderá como ler, processar e analisar CSV de arquivos de texto usando Python. Você verá como os arquivos CSV funcionam, aprenderá sobre a importante [csvbiblioteca](#) integrada ao Python e como funciona a análise de CSV usando a [pandasbiblioteca](#).

O que é um arquivo CSV?

Um arquivo CSV (arquivo de valores separados por vírgula) é um tipo de arquivo de texto simples que usa estruturação específica para organizar dados tabulares. Por ser um arquivo de texto simples, ele pode conter apenas dados de texto reais – em outras palavras, caracteres [ASCII](#) ou [Unicode imprimíveis](#).

A estrutura de um arquivo CSV é revelada pelo seu nome. Normalmente, os arquivos CSV usam vírgula para separar cada valor de dados específico. Esta é a aparência dessa estrutura:

CSV

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

Observe como cada dado é separado por uma vírgula. Normalmente, a primeira linha identifica cada dado – em outras palavras, o nome de uma coluna de dados. Cada linha subsequente são dados reais e são limitadas apenas por restrições de tamanho de arquivo.

Em geral, o caractere separador é chamado de delimitador, e a vírgula não é a única utilizada. Outros delimitadores populares incluem os caracteres tabulação (\t), dois pontos (:) e ponto e vírgula (;). A análise adequada de um arquivo CSV exige que saibamos qual delimitador está sendo usado.

De onde vêm os arquivos CSV?

Os arquivos CSV normalmente são criados por programas que lidam com grandes quantidades de dados. Eles são uma maneira conveniente de exportar dados de planilhas e bancos de dados, bem como importá-los ou usá-los em outros programas. Por exemplo, você pode exportar os resultados de um programa de mineração de dados para um arquivo CSV e depois importá-los para uma planilha para analisar os dados, gerar gráficos para uma apresentação ou preparar um relatório para publicação.

Arquivos CSV são muito fáceis de trabalhar programaticamente. Qualquer linguagem que suporte entrada de arquivo de texto e manipulação de strings (como Python) pode funcionar diretamente com arquivos CSV.

Analizando arquivos CSV com a biblioteca CSV integrada do Python

A `csv` biblioteca fornece funcionalidade para ler e gravar arquivos CSV. Projetado para funcionar imediatamente com arquivos CSV gerados pelo Excel, é facilmente adaptado para funcionar com uma variedade de formatos CSV. A `csv` biblioteca contém objetos e outros códigos para ler, gravar e processar dados de e para arquivos CSV.

Lendo arquivos CSV em um dicionário com CSV

Em vez de lidar com uma lista de elementos individuais , você também pode ler dados CSV diretamente em um dicionário (tecnicamente, um [Dicionário Ordenado](#)).

Novamente, nosso arquivo de entrada `employee_birthday.txt` é o seguinte:

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```



Aqui está o código para lê-lo como um [dicionário](#) desta vez:

Python

```
import csv

with open('employee_birthday.txt', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        print(f'\t{row["name"]} works in the {row["department"]} department, and was l
            line_count += 1
    print(f'Processed {line_count} lines.')
```